Romanian Spelling-Checker

Svetlana Cojocaru, Mikchail Evstiunin and Victor Ufnarovski
Institute of Mathematics of the Moldavian Academy of Sciences
5, Academiei str.,
Kishinev 277028
THE REPUBLIC OF MOLDOVA

Abstract: The implementation of the Romanian Spelling-Checker is discussed. The structure of the vocabulary and similarity word recognition are considered in more detail.

Keywords: natural language processing, lexical databases, computational lexicography and lexicology, grammar.

Svetlana Cojocaru was born in Ribnitsa, The Republic of Moldova. She graduated from the Department of Mathematics and Cybernetics of the Kishinev University in 1974. She obtained her doctoral degree in Computer Science from the Institute of Cybernetics in Kiev (Ukraine) in 1982. Since 1974 she has been working at the Institute of Mathematics of the Academy of Sciences of the Republic of Moldova. Her present position is senior research worker. Dr. Svetlana Cojocaru published more than 30 scientific papers. Her research interests include formal grammars, compiler construction, natural language processing.

Mikchail Evstiunin was born in Korsakov, Russia, in 1964. He graduated from the Department of Mathematics and Cybernetics of the Kishinev University in 1986. He also completed postgraduate courses at the Software Department of the Sankt-Petersburg University (Russia) in 1990. Since 1986 he has been working at the Institute of Mathematics of the Academy of Sciences of The Republic of Moldova. His present position is research worker. Mikchail Evstiunin published more than 10 scientific papers. His research interests include operating systems, compiler construction, natural language processing.

Victor Ufnarovski was born in Kishinev, The Republic of Moldova, in 1955. He graduated from the Department of Mathematics and Mechanics of the Moscow University in 1977. He obtained his doctoral degree in Algebra from the Institute of Mathematics of the USSR Academy of Sciences in 1981. Since 1981 he has been working at the Institute of Mathematics of the Academy of Sciences of The Republic of Moldova. His present position is head of the Software Laboratory. Dr. Victor Ufnarovski published more than 50 scientific papers. His research interests cover computer algebra, compiler construction, natural language processing.

1. Introduction

To create a Spelling-Checker for the Romanian language makes an interesting subject from different points of view. Being one of the highly inflexional languages, the problem of compact representation of its vocabulary is not a trivial one. For example, even for a dictionary of 30 thousand entries, it is necessary that more than 300 thousand different word-forms (the latter will be referred as vocabulary, saving the word "dictionary" for basic word-forms only) should be stored. Taking into account the fact that most of the computers now used in The Republic of Moldova have no more than 1M of RAM, it may be concluded that all the vocabulary (or at least its main corpus) is to be stored on disk. Of course, the problem of getting fast access to the word-forms stored turns up.

Naturally, not only detecting spelling errors, but also being able to suggest correct words is a very topical problem. It would be nice to correct more than one mistake in a word, so, the suggestion becomes a non-trivial problem too.

To formalize the problem, we introduce the notion of binary decomposition of a vocabulary. Special grammars' formalism will be proposed to create a vocabulary.

The related problem of similar words, playing an important part in the suggestion, will be our next subject.

The present approach has been made to the Romanian Spelling-Checker, ROMSP, and some implementation aspects such as increasing efficiency, reducing response time, internal data

representation and architecture, are discussed.

2. The Vocabulary Decomposition and Grammar

Let us suppose a vocabulary V, containing all possible word-forms.

Definition 1 *The binary decomposition of vocabulary V into two sets of words (namely, roots set R and endings set E) and a map f from R to the set of all the subsets of E, satisfying two conditions:*

- *for every root r from R and every ending e from $f(r)$ the concatenation re is the element of V.*
- *for every word v from V a decomposition is possible: a root r from R and an ending e from $f(r)$ such that $v = re$.*

Having V, R, and E, the possible map f may be constructed taking as $f(r)$ the set of all endings e from E such that the concatenation re be in V. One should nevertheless remark, that, as we do not claim an uniqueness of the decomposition of a given word, another map f for the given V, R and E might exist. The problem is how to construct a reasonable map f for recovering V from R and E, with minimum of memory occupied (do remember: the images of roots are sets of words!). Besides, the search time for a given token in the vocabulary should also be reduced .

If V is the word-form vocabulary of a language, there is some hope that taking E and R naturally (in a grammar sense, as they are usually described in manuals, with suffixes included in the roots), the above method leads to a reasonable map. In this case, a list L of all possible values of subsets $f(r)$ will not be so large (if compared with the size of V). So, it will be sufficient to keep on list L, for every root r, only the index of its subset $f(r)$. The memory capacity for the vocabulary will cover two main parts: roots set R memory (every root index memory) and memory of possible sets of endings for list L.

Given the type of decomposition, more than one candidate to the possible roots (or endings) can be found. To avoid ambiguity (and improve access time), some restrictions should be made.

Let V be any set of words.

Definition 2 *Let E be the set of words, including the empty word. The set $R = R_E(V)$, consisting of the minimal left segments r of words w from V, such that the corresponding right segment e ($w = re$) be in E, is named the roots set for V relative to E.*

It goes without saying that on selecting a maximal right segment of w, a possible ending is uniquely determined and only one root checking will do.

We have suggested a decomposition of the vocabulary into two parts. It would be more reasonable to proceed directly on decomposing instead of creating the full vocabulary and selecting roots and endings sets later on. The approach below dawns upon the idea.

Let us consider a grammar rule:

$$[/]^*[\#][N_1]a_1\overline{b_1}a_2\ldots a_{n-1}\overline{b_{n-1}}a_n \longrightarrow a_1'\overline{b_1}a_2'\ldots a_{n-1}'\overline{b_{n-1}}a_n'\ N_2,$$

where a_i , a_i' are arbitrary words and either b_i is a non-empty word or the special symbol $*$ stands for $\overline{b_i}$. N_j — endings set numbers. The interpretation of this rule is as follows.

Let w be the word producing word-forms (basic word-form).

Every sign / indicates dropping the last letter from w. The (after deletions) obtained word v is considered to be a root (if N_1 exists) and N_1- its index on list L of endings sets. In any case, the word v should have the form

$$f_0a_1f_1a_2f_2\ldots a_{n-1}f_{n-1}a_nf_n,$$

where every f_i is arbitrary (possible empty) word, not containing (for $i = 1, 2, \ldots, n-1$) the veto subword b_i. If there exists more than one representation of this kind, the first one (scanning v from the left to the right or vice versa if the sign # is present) should be selected. The special character $*$ instead of $\overline{b_i}$ admits an arbitrary f_i .

After being substituted, the word $f_0a_1'f_1a_2'\ldots a_n'f_n$ serves as a second (or first, if N_1 is absent) root and N_2 is its endings set number.

Note that the special (but frequently occurring) case of this rule

$$[/]^* \longrightarrow N_2,$$

gives the possibility of including v directly.

Using these grammar rules, we can formalize the process of creating the decomposed vocabulary. According to the classification in [1], it is possible that grammar rules for every group are set up. Sometimes more than two roots emerge and more than one grammar rule is needed.

Veto for b_i depends on the position of the subword a_i to be substituted. It seems that for the Romanian language, veto subwords can be avoided (being only restricted by asterisks).

Example 1 *Consider one group of masculine nouns, as described in [1]. One representative word of this group is the word "ied" (a kid). Let us introduce the notations for some morphological features of the Romanian language: N — nominative; A — accusative; G — genitive; D — dative; S — singular, P — plural, V — vocative. Besides, the noun word-forms have two different forms in Romanian: definite form and indefinite form. Let us denote them by FD and FI respectively. Using these notations, we shall list all word-forms for the noun "ied" (the declination faces the alternation "d-z"):*
ied — NASFI or GDSFI,
iezi — NAPFI or GDPFI,
iedul — NASFD,
iedului — GDSFD,
iezii — NAPFD,
iezilor — GDPFD or VP,
iedule — VS.

Here are two roots: "ied" with the endings set:

$$T_1 = \{\text{-, } ul, ului, ule\},$$

and "iez" with the endings set:

$$T_2 = \{i, ii, ilor\},$$

and the grammar rule for the whole group:

$$T_1 d \longrightarrow z\, T_2.$$

Example 2 *Here is the grammar rule for words belonging to one of the most complicated verb groups:*

$$/\#\ T_3\ \breve{a} * \breve{a} \longrightarrow a * \breve{a}\ T_4;$$

$$\#\ \breve{a} * \breve{a} \longrightarrow a * e\ T_5.$$

An equivalent form of this rule is:

$$/T_3\ \breve{a}\bar{\breve{a}}\breve{a}\bar{\breve{a}} \longrightarrow a\bar{\breve{a}}\breve{a}\bar{\breve{a}}\ T_4;$$

$$\breve{a}\bar{\breve{a}}\breve{a}\bar{\breve{a}} \longrightarrow a\bar{\breve{a}}e\bar{\breve{a}}\ T_5.$$

where veto contexts others than the stars were employed.

Three roots are generated for this group (for the sake of brevity, the three corresponding endings sets are overlooked). So, for the verb "dărăpăna" (to collapse), the roots are "dărăpăn", dărapăn", "dărapen".

It is important that the actual list of possible endings sets is not too large: the same ending sets serve for different groups. For example, 100 different groups of masculine nouns use just 15 different sets.

The experiments show that 866 grammar rules and 320 endings sets have been sufficient for the Romanian language to realize the vocabulary decomposition, using 312 endings.

3. Similar Words Detecting

It is well-known that every natural language contains some irregular words (e.g. auxiliary verbs in English or Romanian), which are exceptions to the rules of inflexion. As far as grammar rules constructing is concerned, they are quite unsuitable and sometimes, keeping every word-form separately, instead of constructing rather complicated rules, is to be preferred.

The question is how to determine the degree of irregularity of the word-forms, and (provided that it is not high) how to set up the corresponding grammar rules. Answering this question is sufficient for estimating the similarity of two given words.

The possibility of determining the similarity of two given words may be useful for different purposes. For example, it is helpful to realize the correct word suggestion when the given word has one or more mistakes. But, even in this case, different criteria operate. There may be at least three different causes of mistakes: mistakes arising from a bad knowledge of the word (here similar words are those which sound similarly); mistakes arising from scanner errors (here similar words are those of which letters look similar in a given alphabetic design) and, at last, mistakes arising from wrong key pressing (here similarity depends on the arrangement of the keyboard). As to the grammar rules, similar letters are those which are essential for the

grammatical alternation (see "d" and "z" in Example 1).

The similarity problem has been suggested as liable to a separate two-fold solving. First, the similarity problem-solving for letters only, of which result is a similarity letters matrix M. Given the purpose, this matrix can be filled in independently (in our programs, by integers between 0 and 9).

The second task will be to identify an algorithm capable of calculating the similarity degree of two given words. A common solution will be to assign this function a maximal value if the words differ in at least one letter or in one permutation of two neighbouring letters. In other cases, the function takes zero values. (It can solve the problem of one mistake and be satisfactory for the simplest Spelling-Checker suggestion).

In order to detect more than one mistake (or to set up a grammar rule) more complicated algorithms are to be employed. First of all, let us consider the following variation of the well-known algorithm for searching the maximal common subsequence of the two given words v and w (n and m long correspondingly).

Consider n by m matrix L, where by induction $L[i,j] = \max(L[i-1,j], L[i,j-1], L[i-1,j-1] + M[v[i], w[j]])$, and $L[0,j]$, $L[i,0]$ are considered to be equal to zero. Taking $L[n,m]$ as a similarity criterion, a rather good similarity function (e.g. $f(w,v) = L[n,m]/(n+m)$) is obtained. However, for the suggestion purpose, this method cannot be considered as a satisfactory one: it makes use of too many useless calculations. To improve it, let us restrict our suggestion to a reasonable bound: a similar word should be found if the mistakes made numbered but two. The possibility of correcting three mistakes still remains, but only in the case of "natural" mistakes. This restriction lets us construct only 5 diagonals from $L[i,j]$ matrix (where the difference between i and j is at most 2, $L[i,j]$ being zero in other cases). Besides, it is not necessary to calculate all diagonals: if the values are too small, we should expect more than two mistakes and stop calculations. Actually, the main idea prevailed, even if the real method was slightly more complicated (suggestion should not be too talkative). To accelerate the search, similar roots and similar endings are looked for separately.

One last remark is that that instead of the letter similarity matrix M, the corresponding word similarity tree could be used (it helps, for instance, a more adequate display of the alternation of a letter with several others, which is typical of the Romanian language).

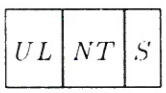
4. Implementation Notes

We will discuss some implementation aspects, unless the morphological information is available. It should be taken into account that two opposite objectives are aimed at:

1. to reduce the database volume;
2. to reduce the access time for achieving reasonable work time.

Obviously, Lempel-Ziv, Lempel-Ziv-Welsh packing methods and their modifications give a good compress coefficient but they do not hold in our case because of a long response time. We have therefore decided on using semantic information during the word packing into a consistent base as only this contains the main words fund, of which volume is huge. Note that compacting the size of database allows it to be scanned more quickly for approaching objective 2.

Consider the representation of a word in the database.



UL — length of a source word root (including the first two letters $C1$ and $C2$)
NT — valid terminations set index
S — common source word root for all terminations in the pointed set (without $C1$ and $C2$)

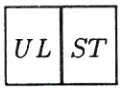
Figure 1: **Consistent Base Page Element Structure**

It can be seen that a word is divided into three parts:

1. the first two characters saved separately (this point will be discussed later);
2. the rest of the root;
3. the set index of valid terminations for this root.

The UL field contains the length of the root in characters including $C1$ and $C2$. The root S is saved in a special encoded form. Thanks to the possibility of calculating the encoded length from the source one (this is irreversible), we use the source words length opposed to the encoded word length.

This field is used for an effective word search. There is no need to compare words when their length is different. It spares a very large string comparison. In case the length and the root are the same, we have to find out whether there is a source word termination in the set pointed out by field NT. The user's base has been subject to an oversimplification. The word is not divided into a root and a termination either.



UL — length of a source word root (including $C1$ and $C2$)
ST — encoded word without $C1$ and $C2$

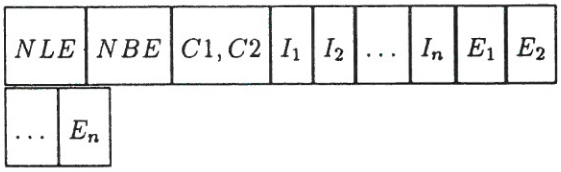
Figure 2: **User's Base Page Element Structure**

The size of the element for the word is limited and consistent. So, the following three cases are possible: the size of a word

1. is equal to the element size;
2. is less than one;
3. is greater than one.

The first case is the ideal case. The second case presents an unused memory. This is not very efficient, but it does not matter after all. The third variant is the most complicated one. The remaining of the word which is not contained in the first element will overlap the next one. So, we must be able to make the distinction between the beginning of a word and its continuation. For this purpose we shall consider the structure of a page.

The first field will show how many roots are stored on the page and the second field will show how many elements are occupied by the roots. Pages as well as elements have a fixed size. So, it is possible that an indexing file should be accessed directly by the page number. Hence we get fast access only by means of an operating system.

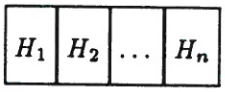


NLE — logical elements quantity
NBE — physical elements quantity
$C1, C2$ — the first two characters of each word on the page
I_i — indirect indexed vector
E_i — page element

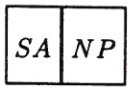
Figure 3: **Consistent and User's Database Page**

The next field keeps the first two characters of all the words saved on the page. This allows for saving some memory without any loss of efficiency. Vector I_i was introduced for simplifying both the database modification and the acquisition of new words. Using this vector is to distinguish between the beginning of a word and its continuation. Using it as an indirect address map sets out the lexicographical order of the elements. A direct element indexing is possible due to a fixed element size. Summarizing the above, we can use binary search methods. Note that such methods allow for both effective searching and words' acquisition. As known, if there are N elements, we can find out the necessary one by log2(N) comparisons. Suppose we have W words in the database and P words on the page, then $D = \log_2 \frac{W}{P} = \log_2 W - \log_2 P$ disk access is needed. Hence, by minimizing parameter D, the time of access to the database is smaller. The following method has been developed. The first word on each page is stored in the special table named HASH-table.

The size of this table will be of D elements. By controlling P we can altogether bring the HASH-table in RAM. Since the pages of the consistent base contain word roots, the volume of pages will grow. A binary search applied to HASH-table makes the page possibly containing the searched word be spotted. Once the page read, we become aware of its containing or not the necessary word. Thus, D is reduced to 1 (i.e. there is only one disk access). The following database general structure is obtained.



H_i — table element



SA — word in ASCII code, the first on the pointed page
NP — page number in the database

Figure 4: **The HASH-table Structure**

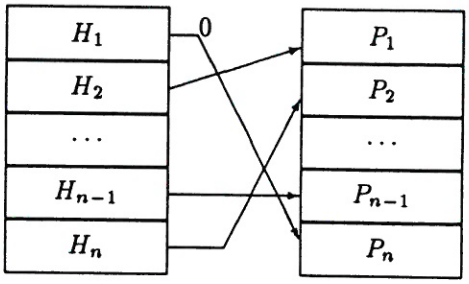


Figure 5: **The General Database Structure**

Efficiency has increased when using the following method. The most frequently used words have been separated in a database under RAM. Its structure is shown in Figure 6.

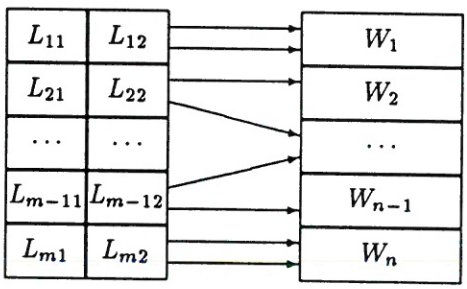


Figure 6: **Database Structure of the Most Frequently Used Words**

Using words length and array L_{ij}, a HASH-function minimizing the operation number of a binary search, results. All words are represented in ASCII code because of the rather small volume of this database.

As a conclusion, we make an evaluation of ROMSP. All measurements have been made on IBM- PC/ AT- compatible computers — 12 MHz, 40 MB HDD, 28 msec average seek.

- Searching speed — 70-100 words per second;
- The volume of the database containing about 330,000 words (3.6 MB) is equal to 700 KB;
- The ROMSP software volume — 300 KB;
- Programming language — TurboPascal.

REFERENCES

1. LOMBARD, A. and GÂDEI, C., **Dictionnaire morphologique de la langue roumaine**, The Academic Publishing House, Bucharest 1981.